FINAL MODELING PROJECT:

PD MODEL WITH DENSITY ESTIMATE TRANSFORMATIONS OF VARIABLES

By: Janul Hernandez and Amir Har-El

Before we start:
We highly recommend looking at the full model posted on GitHub here. All code segments are annotated and explained.

Setup:

You need the following libraries to run the code:

```
#LIBRARIES-----------
library(lubridate)
library(tidyverse)
library(caTools)
```

Our next subsection is the function section. We have several so I will go over them as we get to each one. At the very end of this explanation I will paste the raw functions.

Cleaning Data:

First thing we did was created a new data type called a *tibble*.
A *tibble* is like a dataframe but a little more flexible. Next, we cleaned the dates, stepped forward the report dates by 3 months and created the default flags. The method used was *pipes* from the *tidyverse* package.

```
> bankdata<- bankdata %>%
+   mutate(Default.Date= mdy(Default.Date)) %>%
+   mutate(repdte.adjust= repdte %m+% months(3)) %>%
+   mutate(days.to.default= ifelse (is.na(Default.Date), NA,
+                                   Default.Date-repdte.adjust)) %>%
+   mutate(default.flag= ifelse (is.na(days.to.default), 0, ifelse(days.to.default >0 & days.to.default <= 365.25, 1, 0)))
```

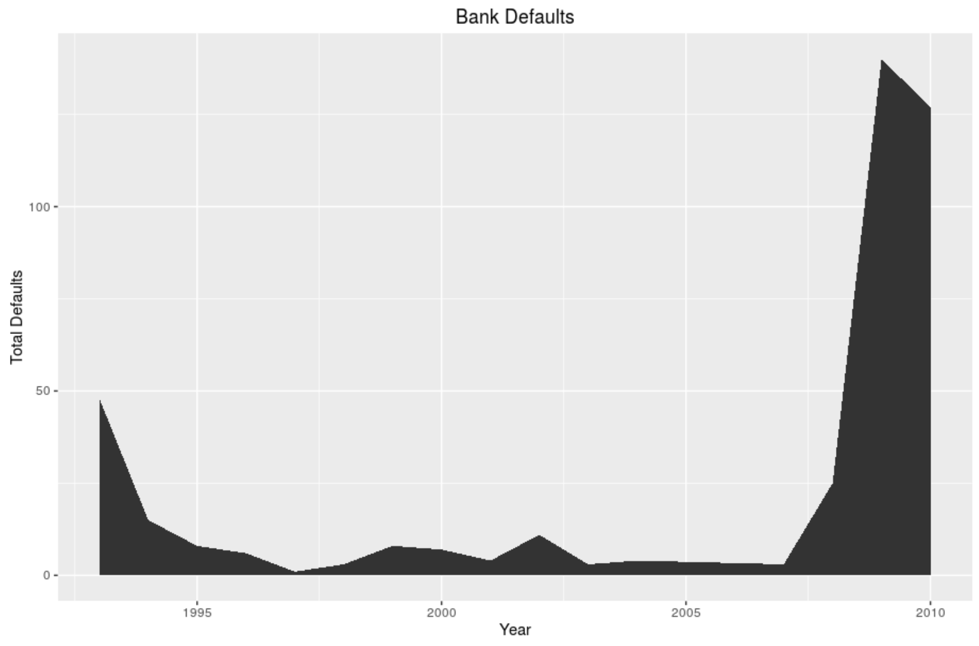
We next created variables we knew we would need for leverage and debt coverage:

```
> bankdata$noi_int<-bankdata$idpretx/(bankdata$eintexp+1)
> bankdata$lev<-bankdata$asset/(bankdata$liab+1)
```

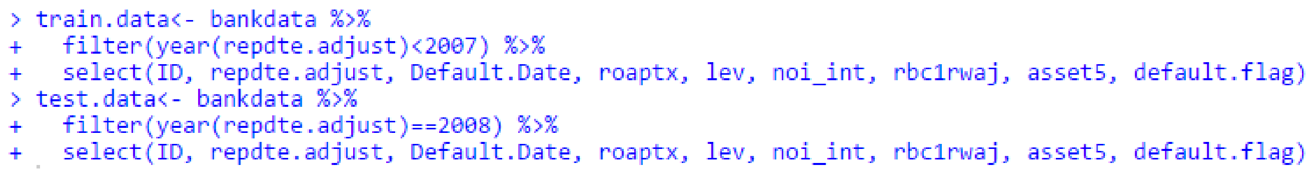
Sampling:

We finally looked at the default timeline with the "*defaultTimeline*()" function:

```
#FUNCTION FOR DEFAULT TIMELINE
defaultTimeline <-function (df)
    #need to send dates as Default.Date and bank ids as ID
{
    output <- df %>%
      filter(!is.na(Default.Date)) %>%
      group_by(yearx= year(Default.Date)) %>%
      summarise(number_banks = n_distinct(ID)) %>%
      ggplot(., aes(x = yearx, y = number_banks)) +
          geom_area() +
          ggtitle("Bank Defaults") +
          labs(x = "Year", y = "Total Defaults")
    return(output)
}
```



From the timeline, we noticed that most of the defaults start in the year 2007 until 2010, which makes sense this was around the financial crisis. We set the training data set to all data prior to 2007 and our test set to 2008. We wanted to check the model power and calibration walking it forward to 2010. One thing to notice is that the following variables are necessary for our model to work (*ID, repdte.adjust, Default.Date, roaptx, lev, noi_int, rbc1rwaj, asset5, default.flag*):



```
> train.data<- bankdata %>%
+   filter(year(repdte.adjust)<2007) %>%
+   select(ID, repdte.adjust, Default.Date, roaptx, lev, noi_int, rbc1rwaj, asset5, default.flag)
> test.data<- bankdata %>%
+   filter(year(repdte.adjust)==2008) %>%
+   select(ID, repdte.adjust, Default.Date, roaptx, lev, noi_int, rbc1rwaj, asset5, default.flag)
```

Variable Transform:

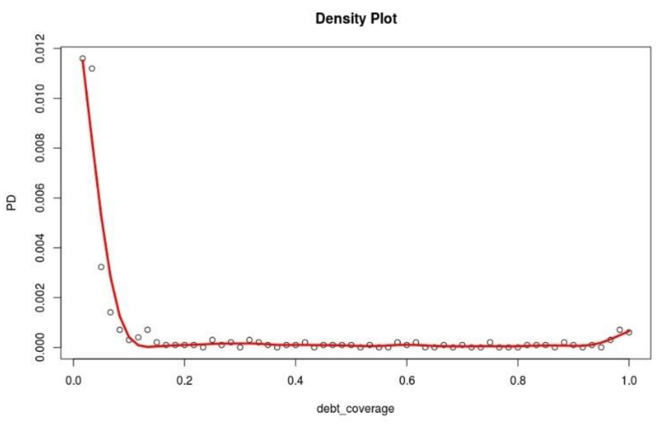
After doing this we transformed our variables using a density estimate of each variable per their respective PD's. We used four functions specifically:

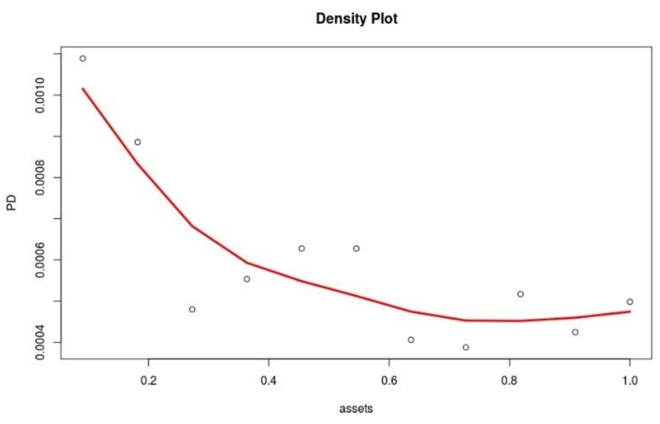
1. *densityMap*()- function takes a vector variable and respective outcome to calculate the PD means and bucket accordingly.
2. *applyDensityMap*()- function takes the map from *densityMap*() and provides appropriate PD's for every variable passed.
3. *transformVar*()- combines *densityMap* and *applyDensityMap* for one function call
4. *autoCal*()- calls *transformVar*() and increments amount of buckets until minimum is no longer negative. This function is good to view the best line fit and then use that bucket value in *transformVar*().

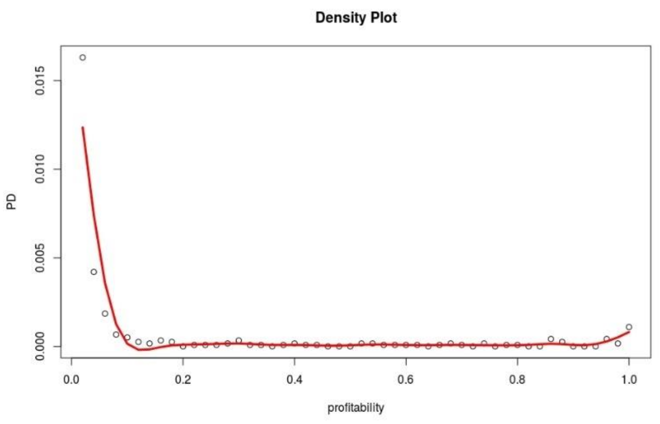
All functions attached to appendix.

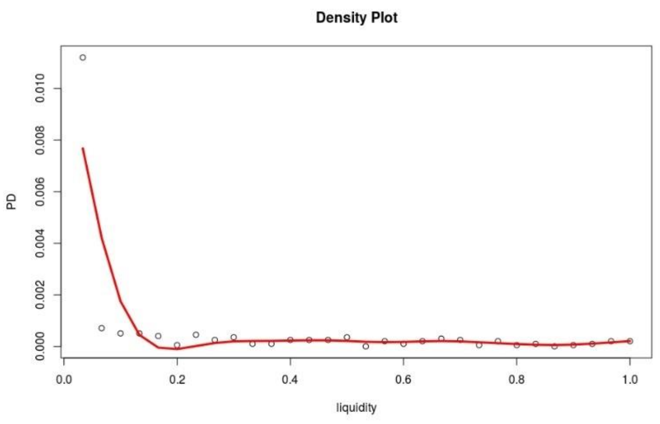
We created density maps for our 5 variables. The following plots are for our initial model. Due to the iterative process of *autoCal*() it would be too many plots to attached on this document. Below you see function calls (please pay close attention to notes after #):

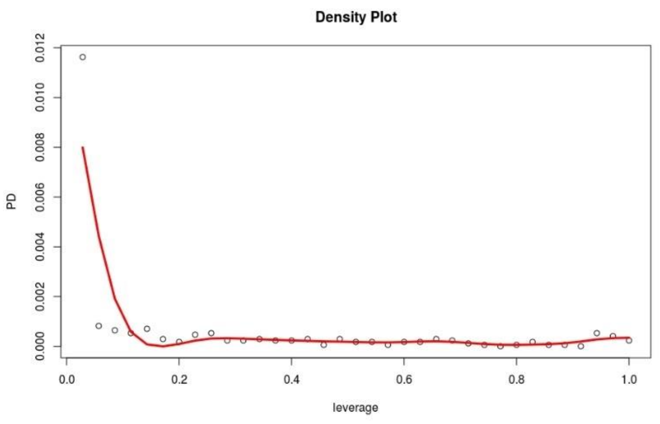
```r
#MEASURE OF PROFITABILITY:
#Pre-tax return on assets- "roaptx"
#We run the autoCal which tries to find the best fit, I chose 50 buckets.
profit.fit<-transformVar(train.data$roaptx,
                        train.data$default.flag,
                        k=50, plot = T, xaxis = "profitability", forceZero = T)

#MEASURE OF LEVERAGE:
#Total asset over total liability- "asset/liability"
#We run the autoCal we find that it solves at k=25 but at k=35 I get more for the curve
leverage.fit<-autoCal(train.data$lev,
                      train.data$default.flag,
                      k=25, plot = T, xaxis = "leverage")

#So we force 0 on any PD below zero with the transformVar forceZero set to true
leverage.fit<-transformVar(train.data$lev,
                           train.data$default.flag,
                           k=35, plot = T, xaxis = "leverage", forceZero = T)

#MEASURES OF DEBT COVERAGE:
#Pre-tax Net operating income over interest expense- "idpretx/eintexp"
#We run autoCal which works extremely well on this datasample. It buckets at 50.
debt.fit<-autoCal(train.data$noi_int,
                  train.data$default.flag,
                  k=60, plot = T, xaxis = "debt_coverage")

#MEASURE OF LIQUIDITY:
#Tier 1 Capital over Total risk-weight assets- "rbc1rwaj"

#We run autoCal and it solves at k=10, but the curve doesn't capture all the information
##note if you start at k=25 autoCal WILL NOT SOLVE due to the shape of the curve
liquidity.fit<- autoCal(train.data$rbc1rwaj,
                        train.data$default.flag,
                        k=20, plot = T, xaxis = "liquidity")

#I force k=30 with forceZero set to true to remove the negative min's is very small
liquidity.fit<- transformVar(train.data$rbc1rwaj,
                             train.data$default.flag,
                             k=30, plot = T, xaxis = "liquidity", forceZero = T)

#MEASURE OF SIZE:
#Average assets- "asset5"
#With more bucketing than 11 the fit stops being monotonically decreasing
#While k=11 is not the best fit, it creates the smoothest line without losing information.
size.fit<-autoCal(train.data$asset5,
                  train.data$default.flag,
                  k=11, plot = T, xaxis = "assets")
```











We next created new columns in our test.data and train.data dataframes with the new fit variables per the mapping above. To do this, we used two functions:

1. *pushTfrm2TRAIN*()- this function takes your raw train.data dataframe and the transform for each variable and adds them to dataframe as the following columns: *debtT*, *levT*, *liqT*, *profT, and sizeT*.
2. *pushTfrm2TEST*()- this function takes your raw test.data mapping from the section above and calculates the new transforms for your data set. It adds the transformed variables as the following columns: *debtT*, *levT*, *liqT*, *profT*, and *sizeT*.

See below for a call of both functions. Note that we pushed the new transformed variables to the train.data and test.data dataframes. We also created a *test.data.clean* dataframe to handle *NA* removal:

```
> train.data<- pushTfrm2TRAIN(train.data, debt.fit, leverage.fit, liquidity.fit, profit.fit, size.fit)
> test.data<- pushTfrm2TEST(test.data, debt.fit, leverage.fit, liquidity.fit, profit.fit, size.fit)
> test.data.clean<- pushTfrm2TEST(test.data, debt.fit, leverage.fit, liquidity.fit, profit.fit, size.fit, naRemove = T)
```

Model Creation and Prediction:

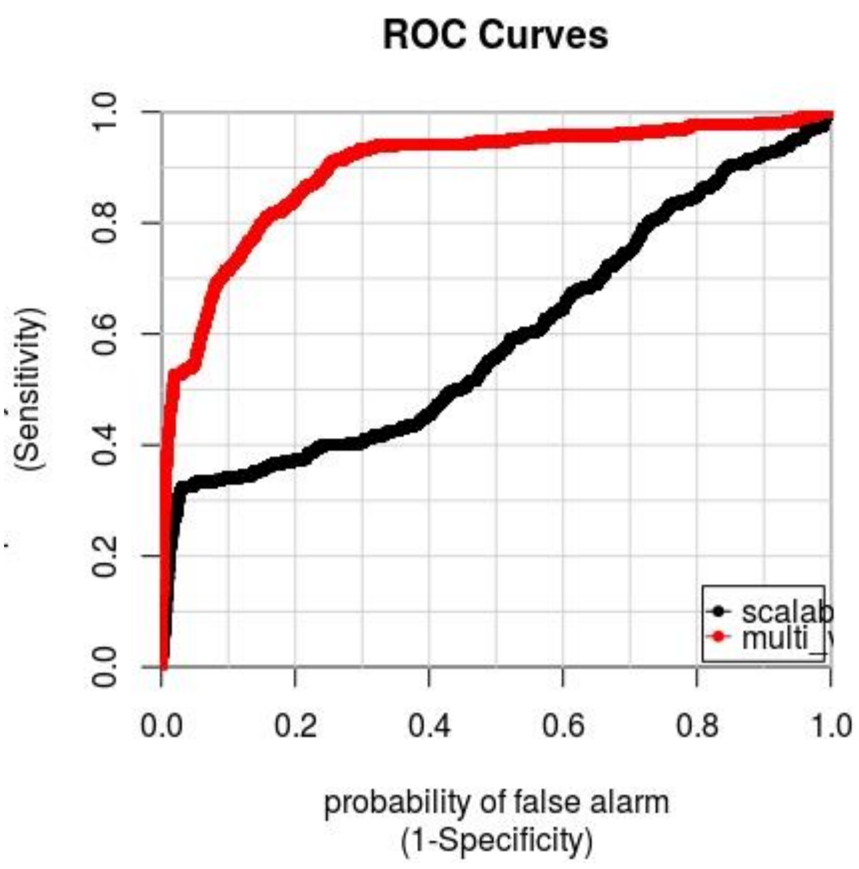
We wanted to create a model that was very flexible and could handle any dataset. The only variables missing NA entries was that of leverage and average asset value, we called this model “scalable”. Next, we created another model with all 5 of the transformed variables called “multi-variate”. From there we created predictions for each model and AUC/ROC plots.

```
#CREATING GLM FOR OUR VARIABLES----------------

#this is our multi-variate model (all 5 are included)
multi.variate<-glm(default.flag ~ debtT + levT + liqT + profT + sizeT,family=binomial(link="logit"),
                    data=train.data, na.action=na.exclude)

predict.multi.variate<-predict(multi.variate , newdat=test.data.clean , type="response",na.action=na.pass)

#this is our scalable model, with the variables (leverage and assets) with no NA's
scalable.model<-glm(default.flag ~ levT + sizeT ,family=binomial(link="logit"),
                    data=train.data, na.action=na.exclude)

predict.scalable<-predict(scalable.model , newdat=test.data.clean , type="response",na.action=na.pass)

#comparing ROC curves we see that our multiVariate model is much more powerfull and accurate than our scalable model
output.Scale<-colAUC(data.frame(scalable=predict.scalable, multi_variate= predict.multi.variate),
                     test.data.clean$default.flag, plotROC=TRUE, alg=c("ROC"))
```

The AUC for the scalable model and multivariate model were:

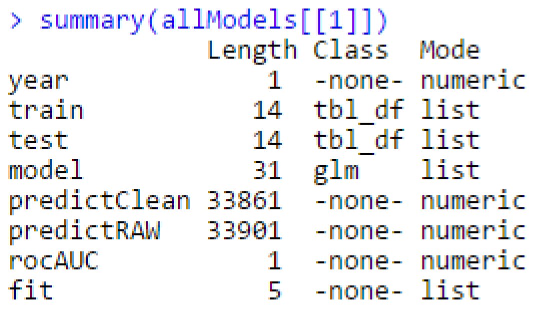
```
> output.Scale
          scalable multi_variate
0 vs. 1 0.5937783     0.8967255
```

The power of our model is again re-enforced by the ROC Curve:



Walk-forward Model Testing:

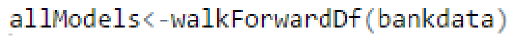
We next wanted to step our model forward from 2005 to 2009 with 1 year hold out test samples. In order to achieve this we created a walk forward function called *walkForwardDf*(). The function carries out all steps above from sampling to model creation and prediction. For each year, the function walks forward it returns a new list labeled by year with the following information:



```
> summary(allModels[[1]])
             Length Class  Mode
year             1  -none- numeric
train           14  tbl_df list
test            14  tbl_df list
model           31  glm    list
predictClean 33861  -none- numeric
predictRAW   33901  -none- numeric
rocAUC           1  -none- numeric
fit              5  -none- list
```

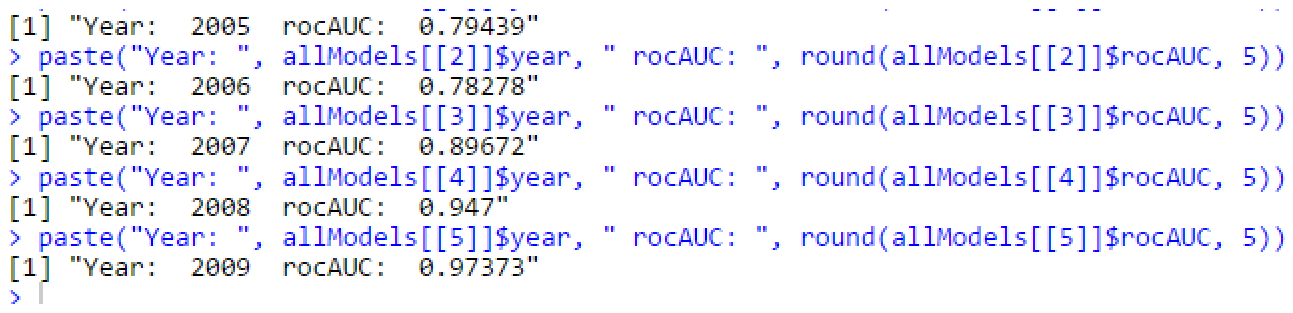
1. *year*- Is a numeric with the year the cutoff occurred
2. *train*- Is a vector of the training dataset including transformed variables
3. *test*- Is the same as above for our test dataset
4. *model*- Is the multi-variate GLM model for the model year
5. *predictClean*- Is the predictions with NA omitted. This vector is used for rocAUC calculation
6. *predictRaw*- Is a vector of predictions for all submissions including NAs
7. *rocAUC*- Is the ROC plot and AUC value
8. *fit*- Is a list of vectors for each fit under our variable transformation

See below for function call:



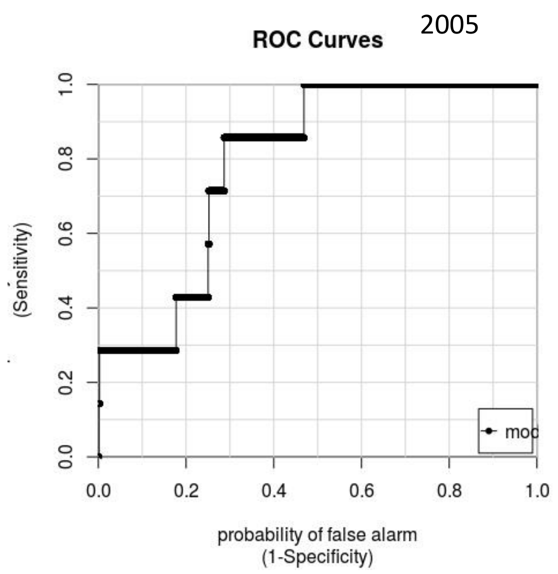
```
allModels<-walkForwardDf(bankdata)
```

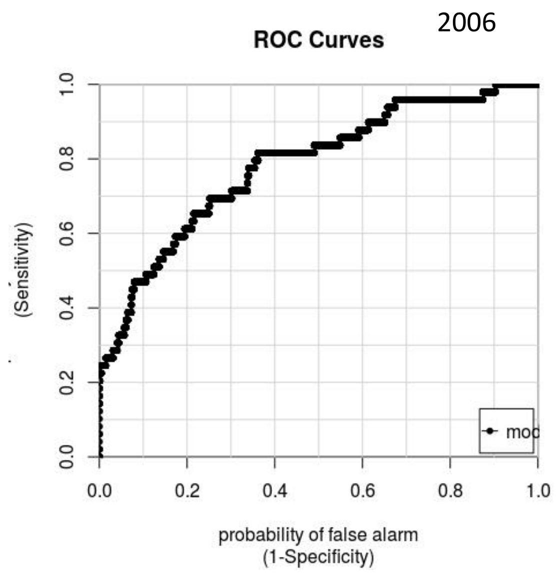
When you call the *walkForwardDf*() function all plots will be generated for density mapping of variables to PDs and ROC curves for each year. Let's see what the function says about our model:

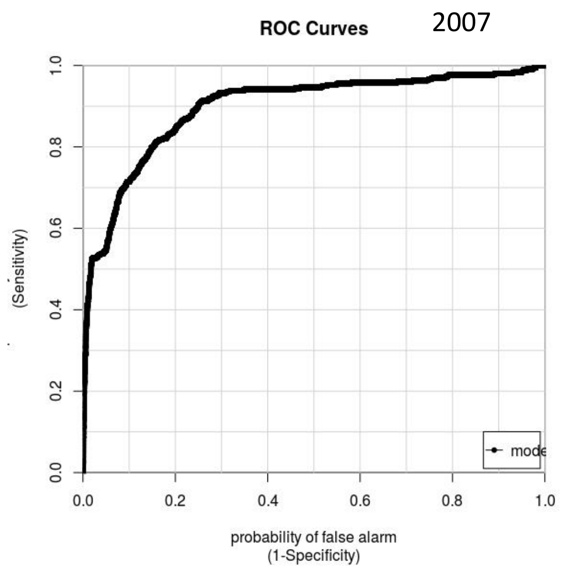


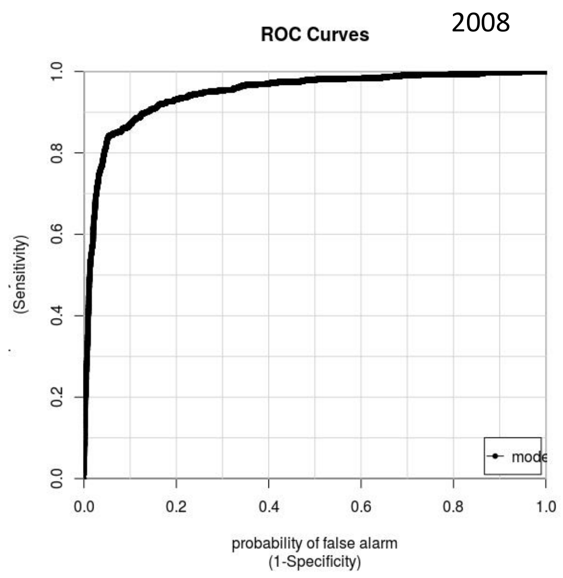
```
[1] "Year:  2005  rocAUC:  0.79439"
> paste("Year: ", allModels[[2]]$year, " rocAUC: ", round(allModels[[2]]$rocAUC, 5))
[1] "Year:  2006  rocAUC:  0.78278"
> paste("Year: ", allModels[[3]]$year, " rocAUC: ", round(allModels[[3]]$rocAUC, 5))
[1] "Year:  2007  rocAUC:  0.89672"
> paste("Year: ", allModels[[4]]$year, " rocAUC: ", round(allModels[[4]]$rocAUC, 5))
[1] "Year:  2008  rocAUC:  0.947"
> paste("Year: ", allModels[[5]]$year, " rocAUC: ", round(allModels[[5]]$rocAUC, 5))
[1] "Year:  2009  rocAUC:  0.97373"
> 
```

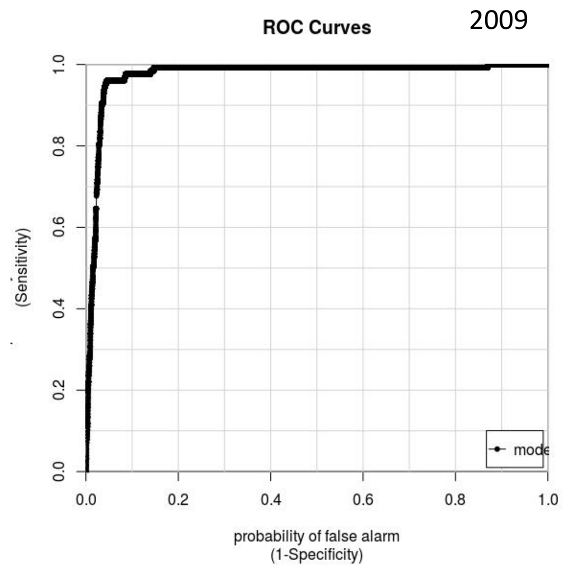
Looks like the AUC for our 2009 model is extremely strong this could be due to the fact a lot of the defaults occurred and our now accounted for in the calibration of our variable transformation. Let's see what the ROC plots look like:


ROC Curves
2005
(Sensitivity)
probability of false alarm
(1-Specificity)
mod

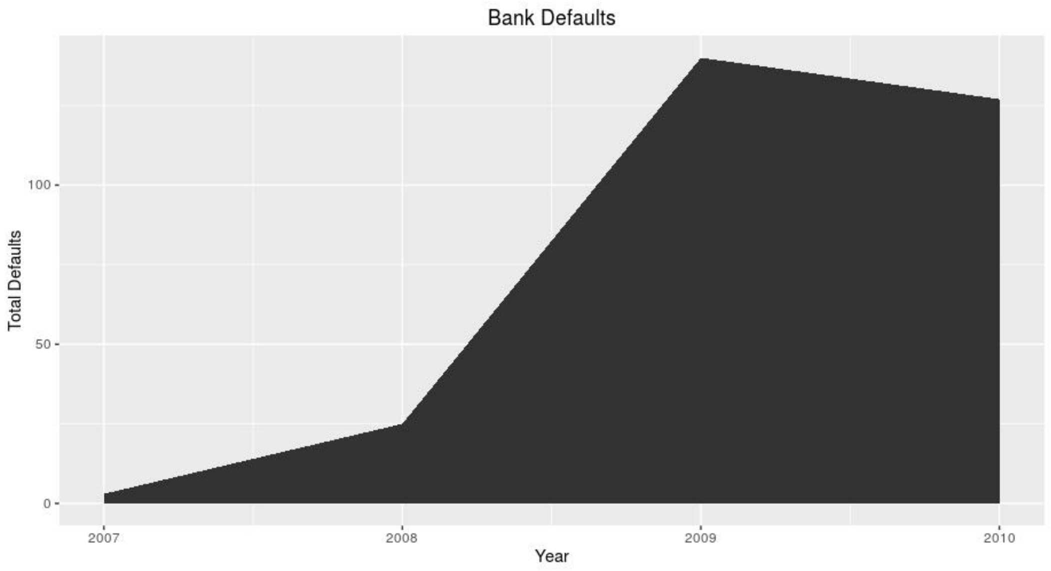

ROC Curves
2006
(Sensitivity)
probability of false alarm
(1-Specificity)
mod

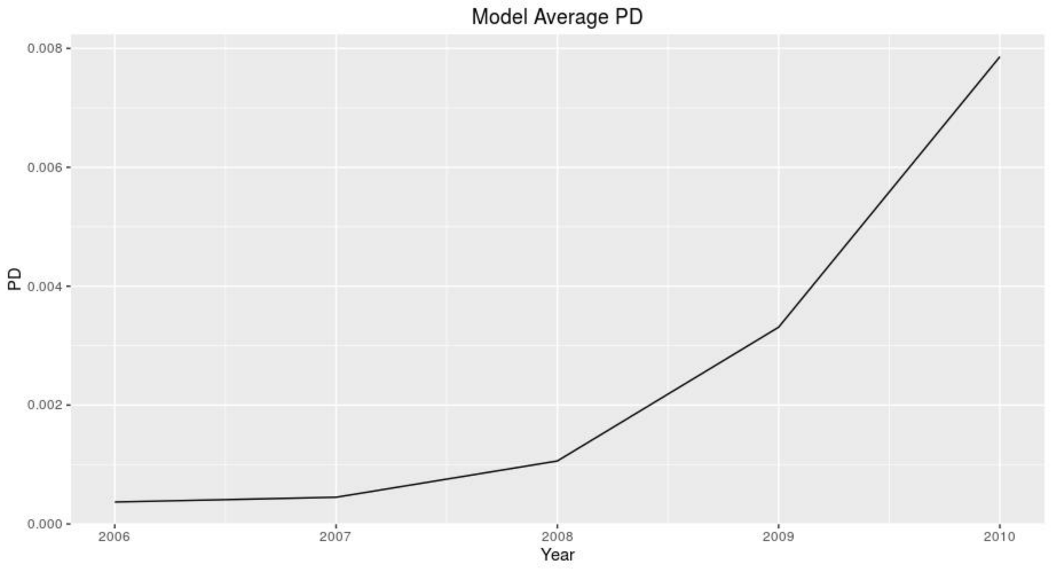

ROC Curves
2007
(Sensitivity)
probability of false alarm
(1-Specificity)
mode


ROC Curves
2008
(Sensitivity)
probability of false alarm
(1-Specificity)
mode


ROC Curves
2009
(Sensitivity)
probability of false alarm
(1-Specificity)
mode

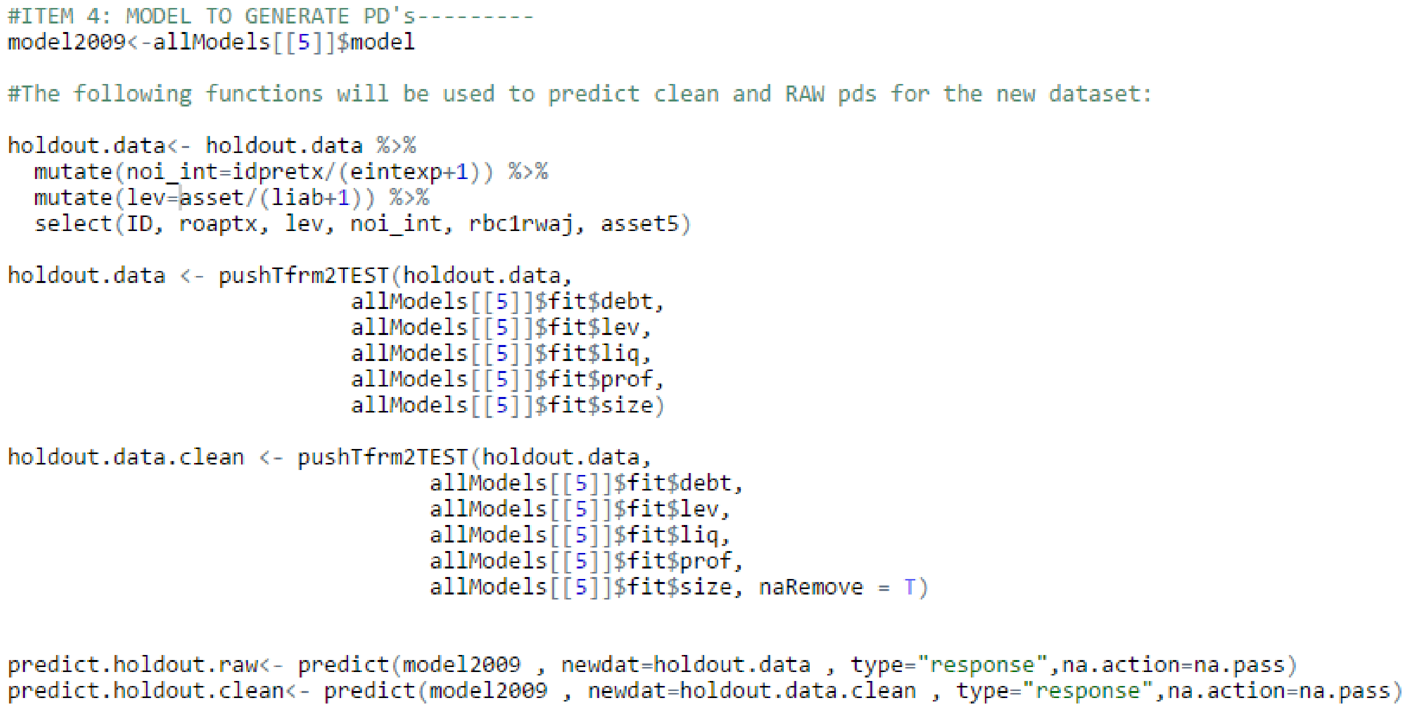
It is also interesting to note the increase in our predicted average PD over time as more banks fail:





Predicting Hold Out Sample (ITEM 4):

With the previous analysis, complete, it is safe to say we will use the 2009 fit model to generate a vector of PD’s with NA excluded and included respectively. The code snippet is below:



```
#ITEM 4: MODEL TO GENERATE PD's---------
model2009<-allModels[[5]]$model

#The following functions will be used to predict clean and RAW pds for the new dataset:

holdout.data<- holdout.data %>%
  mutate(noi_int=idpretx/(eintexp+1)) %>%
  mutate(lev=asset/(liab+1)) %>%
  select(ID, roaptx, lev, noi_int, rbc1rwaj, asset5)

holdout.data <- pushTfrm2TEST(holdout.data,
                          allModels[[5]]$fit$debt,
                          allModels[[5]]$fit$lev,
                          allModels[[5]]$fit$liq,
                          allModels[[5]]$fit$prof,
                          allModels[[5]]$fit$size)

holdout.data.clean <- pushTfrm2TEST(holdout.data,
                                allModels[[5]]$fit$debt,
                                allModels[[5]]$fit$lev,
                                allModels[[5]]$fit$liq,
                                allModels[[5]]$fit$prof,
                                allModels[[5]]$fit$size, naRemove = T)


predict.holdout.raw<- predict(model2009 , newdat=holdout.data , type="response",na.action=na.pass)
predict.holdout.clean<- predict(model2009 , newdat=holdout.data.clean , type="response",na.action=na.pass)
```

Note *holdout.data* is an expected *df* that matches all categories given in the bankdata.new *df*.

FUNCTIONS:

```r
#FUNCTION TO CALCULATE DENSITY ESTIMATES
densityMap  <-  function(x,outcome,k=25,plot=T,xaxis="breaks",spar=0.5)
{
  df =data.frame(x, outcome)
  df<-na.omit(df)

  breaks<-(0:k)/k
  qs<-quantile(df$x, breaks)
  buckets<-cut(df$x, qs)

  raw.curve<-tapply(X=df$outcome , IND=buckets , FUN=mean)
  curve<-smooth.spline(breaks[-1],raw.curve,spar=spar)
  pred<-predict(curve,breaks[-1])

  if(plot)
  {
    plot(breaks[-1],raw.curve,
         main="Density Plot",
         xlab=xaxis,
         ylab="PD")
    lines(pred,col="red",lwd=3)
  }

  baseline<-mean(outcome)
  cutoff<-qs[-1]
  PD<-pred$y
  map<-data.frame(cutoff=cutoff,PD=PD)

  return(list(baseline=baseline,map=map))
}


#FUNCTION TO TRANSFORM VARIABLE TO PD
transformVar <- function(x, outcome, k=25, plot=T, xaxis="breaks", forceZero = F)
{
  fit<- densityMap(x, outcome, k=k, plot=plot, xaxis=xaxis)
  output<- applyDensityMap(x, fit, forceZero)
  return(list(baseline=fit[[1]], map=fit[[2]], transformedVar=output, buckets=k))
}


#AUTO-FIT DENSITY ESTIMATE
autoCal<- function(x, outcome, k=25, plot=T, xaxis="breaks")
{
  output<-transformVar(x , outcome, k , plot , xaxis)

  while(summary(output[[3]])[[1]]<0){
    k=k+5
    output<-transformVar(x , outcome, k , plot , xaxis)
    print(summary(output[[3]])[[1]])
    print(k)
  }
  return(output)
}
```

```r
 #FUNCTION TO USE DENSITY ESTIMATE TO MAP A VARIABLE TO A PD
 applyDensityMap <-  function(x, map, forceZero = F)
 {
   output <- vector("double", length(x))
   cutoff <- map$map$cutoff
   PD <- map$map$PD
   len <- length(cutoff)


   for (i in seq_along(x)) {
     if (is.na(x[i]))
       pd <- NA
     else{
       lb <- sum(cutoff < x[i])
       if (lb == 0)
         pd <- PD[1]
       else  if (lb >= len)
         pd <- PD[len]
       else
       {
         ub <- lb + 1
         num <- x[i] - cutoff[lb]
         den <- cutoff[ub] - cutoff[lb]
         wt.ub <- num / den
         wt.lb <- 1 - wt.ub
         pd<-wt.lb*PD[lb]+wt.ub*PD[ub]
       }
     }


     if (forceZero){
       if (pd>0 || is.na(pd)) output[i]<- pd
       else output[i]<- 0
     }
     else{output[i]<- pd}
   }
   return(output)
 }

#FUNCTION TO PUSH TRANFORM VARIABLES TO TRAIN df
pushTfrm2TRAIN<- function(df, debt, lev, liq, prof, size){
  df$debtT<- debt[[3]]
  df$levT<- lev[[3]]
  df$liqT<- liq[[3]]
  df$profT<- prof[[3]]
  df$sizeT<- size[[3]]

  return(df)
}

#FUNCTION TO TRANSFORM VARIBLES TO TEST df
pushTfrm2TEST<- function(df, debt, lev, liq, prof, size, naRemove= F){
  df$debtT<- applyDensityMap(df$noi_int ,debt, forceZero = T)
  df$levT<- applyDensityMap(df$lev ,lev, forceZero = T)
  df$liqT<- applyDensityMap(df$rbc1rwaj ,liq, forceZero = T)
  df$profT<- applyDensityMap(df$roaptx ,prof, forceZero = T)
  df$sizeT<- applyDensityMap(df$asset5 ,size, forceZero = T)

  if(naRemove){
    df<- df %>%
      filter(!is.na(profT)) %>%
      filter(!is.na(rbc1rwaj))
  }

  return(df)
}
```

```r
#THE WALKFORWARD FUNCTION: SEND THE DATAFRAME IT DOES THE REST
#THIS FUNCTION CALCULATES NEW FITS ACCORDING TO ADDITIONAL DATA,
#REMOVES ALL NA's FROM TEST SET
#RETURNS A LIST OF (YEAR, MODEL, TEST DATA, TRAIN DATA, RAW PREDICT,
#CLEAN PREDICT, ROC FOR MODEL)
walkForwardDf<- function(df, yearX=2007, yearOutofSample=1){

  i=1
  output<-list()

  while(yearX<year(max(df$repdte.adjust))){

    train.data<- df %>%
      filter(year(repdte.adjust)<yearX) %>%
      select(ID, repdte.adjust, Default.Date, roaptx, lev, noi_int, rbc1rwaj, asset5, default.flag)

    test.data<- df %>%
      filter(year(repdte.adjust)==(yearX+yearOutofSample)) %>%
      select(ID, repdte.adjust, Default.Date, roaptx, lev, noi_int, rbc1rwaj, asset5, default.flag)

    ##calculating new fits
    #PROFIT
    profit.fit<-transformVar(train.data$roaptx,
                             train.data$default.flag,
                             k=50, plot = F, xaxis = "roaptx")

    #MEASURE OF LEVERAGE:
    leverage.fit<-transformVar(train.data$lev,
                               train.data$default.flag,
                               k=35, plot = T, xaxis = "leverage", forceZero = T)

    #MEASURES OF DEBT COVERAGE:
    debt.fit<-transformVar(train.data$noi_int,
                           train.data$default.flag,
                           k=60, plot = T, xaxis = "debt_coverage")

    #MEASURE OF LIQUIDITY:
    liquidity.fit<- transformVar(train.data$rbc1rwaj,
                                 train.data$default.flag,
                                 k=30, plot = T, xaxis = "liquidity", forceZero = T)

    #MEASURE OF SIZE:
    size.fit<-transformVar(train.data$asset5,
                           train.data$default.flag,
                           k=11, plot = T, xaxis = "assets")

    #TRANSFORMING VARIABLES
    train.data<- pushTfrm2TRAIN(train.data, debt.fit, leverage.fit, liquidity.fit, profit.fit, size.fit)
    test.data.RAW<- pushTfrm2TEST(test.data, debt.fit, leverage.fit, liquidity.fit, profit.fit, size.fit, naRemove = F)
    test.data.clean<- pushTfrm2TEST(test.data, debt.fit, leverage.fit, liquidity.fit, profit.fit, size.fit, naRemove = T)

    #CREATING NEW MODEL AND CALCULATIONS
    model<-glm(default.flag ~ debtT + levT + liqT + profT + sizeT,family=binomial(link="logit"),
                   data=train.data, na.action=na.exclude)

    #Clean dataset passed, no NA's the length of the predict vector may differ
    predict.clean<-predict(model , newdat = test.data.clean , type="response",na.action=na.pass)

    #RAW dataset all NA's passed, this can't be used to calculate ROC due to NA inclusion.
    predict.raw<-predict(model , newdat = test.data.RAW , type="response",na.action=na.pass)

    rocAUC<-colAUC(data.frame(model= predict.clean), test.data.clean$default.flag, plotROC=TRUE, alg=c("ROC"))

    #step forward in list and return year, training data, test data, model, predictClean, predictRAW
    output[[i]]<-list(year=yearX, train=train.data, test=test.data.clean, model= model, predictClean=predict.clean,
                      predictRAW=predict.raw, rocAUC= rocAUC,
                      fit= list(prof=profit.fit,lev=leverage.fit, debt=debt.fit, size=size.fit, liq=liquidity.fit))

    yearX=yearX+yearOutofSample
    i= i+1
  }
  return(output)
}
```